

# APPLICATION FOR U.S. PATENT

## INHERITABLE THREAD-LOCAL STORAGE

INVENTORS: Joshua J. Bloch  
1199 Cordelia Avenue  
San Jose, CA 95129

A Citizen of the United States of America

ASSIGNEE: SUN MICROSYSTEMS, INC.  
901 SAN ANTONIO ROAD  
PALO ALTO, CA 94303

A DELAWARE CORPORATION

ENTITY: LARGE

BEYER & WEAVER, LLP  
P.O. Box 61059  
Palo Alto, CA 94306  
Telephone (650) 493-2100

SDB/DBJ

# INHERITABLE THREAD-LOCAL STORAGE

## BACKGROUND OF THE INVENTION

### 1. Field of the Invention

The present invention relates generally to the field of computer software, and more particularly to inheritable thread-local storage.

### 2. Description of the Related Art

Modern multitasking operating systems support multiple threads, which allow many activities to execute simultaneously. For example, searches on multiple remote databases may all be launched by one task, each search corresponding to a unique thread, and executed simultaneously. Oftentimes, it is convenient to have storage that is unique to each thread.

Unfortunately, most programming languages do not support such storage. However, there is a standard software mechanism for threads to associate implicit scope information with themselves, known variously as “thread-local data” or “thread-specific data.” This facility exists in many threading facilities, including POSIX Threads (“Pthreads”) and Win32 Threads.

Under POSIX, thread-specific data allows each thread to have a separate copy of a variable, which is indexed by a common “key” value. A key is created and each thread independently sets or gets its own unique value for that key. The key is the same for all the threads, but each thread can associate its own unique value with the shared key. Each thread can change its private value for a key without affecting the key or any other thread’s value for the key. When the threading system is asked for a new key, a new variable is created. This type of “key” system is not secure, however, since the keys can be forged and data accessed without permission. The POSIX thread-specific data mechanism is explained in further detail in *Programming in POSIX Threads*, David R. Butenhof, Addison-Wesley, 1997, herein

incorporated by reference. A similar mechanism is available in the Windows 95 operating system, and is discussed in *Programming Windows 95*, Charles Petzold, Microsoft Press, 1996, herein incorporated by reference.

The Java <sup>TM</sup> programming language, created by Sun Microsystems, is somewhat unique among programming languages in that it has built-in thread-local storage support. Specifically, there is a public class “ThreadLocal” that provides thread-local variables. The thread-local variables differ from the other “normal” Java variables in that each thread that accesses a thread-local variable (via a get or set method) has its own, independently initialized copy of the variable. ThreadLocal objects are typically private static variables in classes that want to associate a state with a thread, such as a user ID or a transaction ID. Each thread holds an implicit reference to its copy of a ThreadLocal as long as the thread is in existence and is accessible. After a thread terminates, all of its copies of ThreadLocal variables are subject to garbage collection, unless other references to these copies exist. Figure 3 is a table illustrating the API specification for the class ThreadLocal.

For many programming applications, is often desirable for threads to carry with them some implicit “scope” information, such as the principal or transaction on whose behalf the thread is executing. When a thread (“the parent”) creates another thread (“the child”), it is often desirable that this scope information be automatically transmitted from parent to child, in a manner that may depend on the details of the scope information in question. The thread-specific data (hereinafter “thread-local data” or “thread-local storage”) mechanisms described above do not solve this problem, however, since values are not passed from parent threads to child threads.

In view of the foregoing, it would be desirable to have a thread-local storage mechanism in which a child thread can inherit its parent's values directly, or in which a child thread could inherit values which are some functions of the parent's values.

### **SUMMARY OF THE INVENTION**

5 In a threading mechanism, the present invention is a system and method for providing automatic value inheritance when a parent thread creates a child thread. Upon the creation of a child thread, the system iterates over all of the inheritable thread-local values associated with the parent thread and initializes the child's values of these inheritable thread-local values, based on an appropriate childValue method.

0 In a first embodiment, for each thread, a hash table maps each thread local object to a value. In a preferred implementation, the hash table is two separate logical maps - one for inheritable values and one for non-inheritable values. When a thread creates a child, the system iterates over the inheritable value map to create the child's values. An inheritance protocol (i.e. the "childValue" method) may be performed on the values in order to calculate the child's value as a function of the parent's value, if desired.

5 In an alternative embodiment, the two hash tables in the first embodiment can be combined into one table, with each entry having a flag to identify the inheritable values.

20 In a second embodiment of the present invention, for each thread-local variable, a hash table maps each thread to a value. An object reference to a current thread is used as a look-up key in the hash table to find the value associated with this thread. For each thread, a linked list called "values" links all the inheritable thread-local values associated with the thread. The head pointer to the linked list of inheritable values is stored in the thread object. When a parent thread creates a child thread, the system iterates over the linked list of thread-local values

pertaining to the parent thread. For each parent value, a “childValue” method is invoked to initialize the associated child’s value.

In a preferred embodiment, the present invention is implemented in the Java™ programming language. The InheritableThreadLocal class of the present invention extends the ThreadLocal class to provide inheritance of values from a parent thread to a child thread. When a child thread is created, the child receives initial values for all InheritableThreadLocals for which the parent has values. Normally, the child’s values will be identical to the parent’s values. However, the child’s value can be made to be an arbitrary function of the parent’s value by overriding the “childValue” method in the InheritableThreadLocal class.

### **BRIEF DESCRIPTION OF THE DRAWINGS**

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

Figure 1 is a block diagram of a computer system suitable for implementing the present invention;

Figure 2 is a block diagram illustrating value inheritance for a child thread;

Figure 3 is table of a prior art ThreadLocal API specification;

Figure 4 is a table of the InheritableThreadLocal API specification of the present invention;

Figure 5 is a diagram illustrating a first embodiment of the present invention;

Figure 6 is a diagram of an alternative implementation of the first embodiment of the present invention; and

Figure 7 is a diagram illustrating a second embodiment of the present invention.

## **DETAILED DESCRIPTION OF THE INVENTION**

The following description is provided to enable any person skilled in the art to make and use the invention and sets forth the best modes contemplated by the inventor for carrying out the invention. Various modifications, however, will remain readily apparent to those skilled in the art, since the basic principles of the present invention have been defined herein specifically to provide an inheritable thread-local storage mechanism.

The present invention employs various computer-implemented operations involving data stored in computer systems. These operations include, but are not limited to, those requiring physical manipulation of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. The operations described herein that form part of the invention are useful machine operations. The manipulations performed are often referred to in terms, such as, producing, identifying, running, determining, comparing, executing, downloading, or detecting. It is sometimes convenient, principally for reasons of common usage, to refer to these electrical or magnetic signals as bits, values, elements, variables, characters, data, or the like. It should be remembered, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities.

The present invention also relates to a device, system or apparatus for performing the aforementioned operations. The system may be specially constructed for the required purposes, or it may be a general purpose computer selectively activated or configured by a computer program stored in the computer. The processes presented above are not inherently related to any particular computer or other computing apparatus. In particular, various

general-purpose computers may be used with programs written in accordance with the teachings herein, or, alternatively, it may be more convenient to construct a more specialized computer system to perform the required operations.

FIG. 1 is a block diagram of a general purpose computer system 100 suitable for carrying out the processing in accordance with one embodiment of the present invention. Figure 1 illustrates one embodiment of a general purpose computer system. Other computer system architectures and configurations can be used for carrying out the processing of the present invention. Computer system 100, made up of various subsystems described below, includes at least one microprocessor subsystem (also referred to as a central processing unit, or CPU) 102. That is, CPU 102 can be implemented by a single-chip processor or by multiple processors. It should be noted that in re-configurable computing systems, CPU 102 can be distributed amongst a group of programmable logic devices. In such a system, the programmable logic devices can be reconfigured as needed to control the operation of computer system 100. In this way, the manipulation of input data is distributed amongst the group of programmable logic devices. CPU 102 is a general purpose digital processor which controls the operation of the computer system 100. Using instructions retrieved from memory, the CPU 102 controls the reception and manipulation of input data, and the output and display of data on output devices.

CPU 102 is coupled bi-directionally with a first primary storage 104, typically a random access memory (RAM), and uni-directionally with a second primary storage area 106, typically a read-only memory (ROM), via a memory bus 108. As is well known in the art, primary storage 104 can be used as a general storage area and as scratch-pad memory, and can also be used to store input data and processed data. It can also store programming instructions and data, in the form of data objects, in addition to other data and instructions for

processes operating on CPU 102, and is used typically used for fast transfer of data and instructions in a bi-directional manner over the memory bus 108. Also as well known in the art, primary storage 106 typically includes basic operating instructions, program code, data and objects used by the CPU 102 to perform its functions. Primary storage devices 104 and 106 may include any suitable computer-readable storage media, described below, depending on whether, for example, data access needs to be bi-directional or uni-directional. CPU 102 can also directly and very rapidly retrieve and store frequently needed data in a cache memory 110.

A removable mass storage device 112 provides additional data storage capacity for the computer system 100, and is coupled either bi-directionally or uni-directionally to CPU 102 via a peripheral bus 114. For example, a specific removable mass storage device commonly known as a CD-ROM typically passes data uni-directionally to the CPU 102, whereas a floppy disk can pass data bi-directionally to the CPU 102. Storage 112 may also include computer-readable media such as magnetic tape, flash memory, signals embodied on a carrier wave, PC-CARDS, portable mass storage devices, holographic storage devices, and other storage devices. A fixed mass storage 116 also provides additional data storage capacity and is coupled bi-directionally to CPU 102 via peripheral bus 114. The most common example of mass storage 116 is a hard disk drive. Generally, access to these media is slower than access to primary storages 104 and 106.

Mass storage 112 and 116 generally store additional programming instructions, data, and the like that typically are not in active use by the CPU 102. It will be appreciated that the information retained within mass storage 112 and 116 may be incorporated, if needed, in standard fashion as part of primary storage 104 (e.g. RAM) as virtual memory.



In addition to providing CPU 102 access to storage subsystems, the peripheral bus 114 is used to provide access other subsystems and devices as well. In the described embodiment, these include a display monitor 118 and adapter 120, a printer device 122, a network interface 124, an auxiliary input/output device interface 126, a sound card 128 and speakers 130, and other subsystems as needed.

The network interface 124 allows CPU 102 to be coupled to another computer, computer network, or telecommunications network using a network connection as shown. Through the network interface 124, it is contemplated that the CPU 102 might receive information, *e.g.*, data objects or program instructions, from another network, or might output information to another network in the course of performing the above-described method steps. Information, often represented as a sequence of instructions to be executed on a CPU, may be received from and outputted to another network, for example, in the form of a computer data signal embodied in a carrier wave. An interface card or similar device and appropriate software implemented by CPU 102 can be used to connect the computer system 100 to an external network and transfer data according to standard protocols. That is, method embodiments of the present invention may execute solely upon CPU 102, or may be performed across a network such as the Internet, intranet networks, or local area networks, in conjunction with a remote CPU that shares a portion of the processing. Additional mass storage devices (not shown) may also be connected to CPU 102 through network interface 124.

Auxiliary I/O device interface 126 represents general and customized interfaces that allow the CPU 102 to send and, more typically, receive data from other devices such as microphones, touch-sensitive displays, transducer card readers, tape readers, voice or

handwriting recognizers, biometrics readers, cameras, portable mass storage devices, and other computers.

Also coupled to the CPU 102 is a keyboard controller 132 via a local bus 134 for receiving input from a keyboard 136 or a pointer device 138, and sending decoded symbols from the keyboard 136 or pointer device 138 to the CPU 102. The pointer device may be a mouse, stylus, track ball, or tablet, and is useful for interacting with a graphical user interface.

In addition, embodiments of the present invention further relate to computer storage products with a computer readable medium that contain program code for performing various computer-implemented operations. The computer-readable medium is any data storage device that can store data which can thereafter be read by a computer system. The media and program code may be those specially designed and constructed for the purposes of the present invention, or they may be of the kind well known to those of ordinary skill in the computer software arts. Examples of computer-readable media include, but are not limited to, all the media mentioned above: magnetic media such as hard disks, floppy disks, and magnetic tape; optical media such as CD-ROM disks; magneto-optical media such as floptical disks; and specially configured hardware devices such as application-specific integrated circuits (ASICs), programmable logic devices (PLDs), and ROM and RAM devices. The computer-readable medium can also be distributed as a data signal embodied in a carrier wave over a network of coupled computer systems so that the computer-readable code is stored and executed in a distributed fashion. Examples of program code include both machine code, as produced, for example, by a compiler, or files containing higher level code that may be executed using an interpreter.

It will be appreciated by those skilled in the art that the above described hardware and software elements are of standard design and construction. Other computer systems suitable

for use with the invention may include additional or fewer subsystems. In addition, memory bus 108, peripheral bus 114, and local bus 134 are illustrative of any interconnection scheme serving to link the subsystems. For example, a local bus could be used to connect the CPU to fixed mass storage 116 and display adapter 120. The computer system shown in FIG. 1 is but an example of a computer system suitable for use with the invention. Other computer architectures having different configurations of subsystems may also be utilized.

In general, the inheritable thread-local storage of the present invention is similar to ordinary thread-local storage, available in most modern threading facilities (such as POSIX Threads or Win32 Threads). However, the inheritable thread-local storage values of the present invention are automatically passed from a parent thread to a child thread (“inherited”) when a child thread is created, as shown in Figure 2. In Figure 2, Thread 1 creates a new child, Thread 1.1. Thread 1’s value 1A is passed via the childValue method to the child Thread 1.1 as value 1.1A. Similarly, all other inheritable values associated with Thread 1 are transferred to Thread 1.1. Thus, the child Thread 1.1 receives initial values for all inheritable thread-local variables for which the parent Thread 1 has values. Similarly, Thread 2 creates a new child, Thread 2.2, having an inherited value 2.1A.

The present invention is a general-purpose facility that overcomes the inheritance limitations of the prior art systems. Although the present invention is described with reference to a preferred embodiment in the Java programming language, the teachings of the present invention are applicable to any implementation supporting thread-local storage. The teachings or the present invention are particularly applicable to other programming languages that support threads.

In the presently preferred embodiment, the present invention is implemented in the Java programming language. By using Java, the present invention is able to take advantage of the

data security inherent in the language. The present implementation does not use “keys,” but instead uses actual Java object references. An InheritableThreadLocal class API specification of the present invention is illustrated in the table of Figure 4. When an object of type ThreadLocal or InheritableThreadLocal is created, only someone with access to the object is allowed to read the variables, thus providing improved data security, as compared to the “key” based systems.

The InheritableThreadLocal class of the present invention extends ThreadLocal to provide inheritance of values from a parent thread to a child thread. When a child thread is created, the child receives initial values for all InheritableThreadLocals for which the parent has values. Normally, the child’s values will be identical to the parent’s values. However, the child’s value can be made to be an arbitrary function of the parent’s value by overriding the “childValue” method in the InheritableThreadLocal class.

In general, the present invention may be implemented by either mapping, for each thread-local variable, each thread to a value, or for each thread, mapping each thread-local variable to a value. Both embodiments are discussed in detail below.

In a first embodiment, for each thread, a hash table maps each thread-local variable to a value, as shown in Figure 5. More particularly, under Java, each thread-local variable is represented by a thread-local reference in the hash table and the thread-local reference is mapped to a particular value. In a preferred implementation, separate hash tables are maintained for each logical map - one for inheritable values and one for non-inheritable values.

When a thread creates a child, the system iterates over the inheritable value map to create the child’s values. An inheritance protocol can then be performed on the values in order to calculate the child’s value as a function of the parent’s value, if necessary. This implementation requires only one data structure, since no linked list is necessary. Also, since the map is only accessed by the one thread associated with the values, no synchronization is required. In a

second embodiment, described below, the maps are shared so a synchronization scheme must be implemented to ensure that the values are not inadvertently overwritten by another thread.

In an alternative embodiment, the two hash tables in the first embodiment can be combined as shown in Figure 6, with each entry having a logical flag to identify the inheritable values. As in the first embodiment, when a thread creates a child, the system iterates over the inheritable value to create the child's values.

When a parent thread creates a child thread, the system iterates over the inheritable thread-local values pertaining to the parent thread. For each parent value, a "childValue" method is invoked to initialize the associated child's value. The default childValue method may be overridden to compute a desired value as a function of the parent value. If the childValue method has not been overridden, then by default, the value is just a copy of the parent's value. Thus, a child thread "inherits" the values (or some function of the values) of the parent thread.

In a second embodiment of the present invention, for each thread-local variable, a hash table maps each thread to a value, as illustrated in Figure 7. An object reference to a current thread is used as a look-up key in the hash table to find the value associated with this thread. For each thread, a linked list called "values" links all the inheritable thread-local values associated with the thread. The head pointer to the linked list of inheritable values is stored in the thread object. Note that for the non-inheritable values, a linked list is not necessary since the child thread will not inherit these values.

When a parent thread creates a child thread, the system iterates over the linked list of thread-local values pertaining to the parent thread. For each parent value, a "childValue" method is invoked to initialize the associated child's value. The default childValue method may be overridden to compute a desired value as a function of the parent value. If the childValue

method has not been overridden, then by default, the value is just a copy of the parent's value.

Thus, a child thread "inherits" the values (or some function of the values) of the parent thread.

Further, in the second embodiment the hash table should contain only "weak references" to the threads, to allow the threads (and their thread-local values) to be garbage collected when they terminate. In a preferred implementation, the hash tables are implemented as "WeakHashMap" objects in Java.

Inheritable thread-local storage is used in preference to ordinary thread-local storage when the per-thread-attribute being maintained in the storage must be automatically transmitted to any child threads that are created. Common uses include such attributes as the user ID or transaction ID on whose behalf a computation is taking place. Storing such an attribute in inheritable thread-local storage automatically turns child threads into "natural extensions" of their parent, operating on behalf of the same entity.

The more general form of inheritable thread-local storage (where the child's initial value is an arbitrary function of the parent's) enables attributes where the child's initial value for the attribute is not identical to the parent's, but is derivable from it. For example, consider a dynamically scoped attribute, such as a transaction ID in a system that supports nested transactions. For such an attribute, each thread maintains in inheritable thread-local storage a "stack" of values representing the nested scopes currently in effect. The child's initial value for such a thread-local variable should be a stack with one element, that is, the top of the parent's stack.

Thus, according to the present invention, upon thread creation, the system iterates over all of the inheritable thread-local values associated with a parent thread and initializes the child's values of these inheritable thread-local values based on the appropriate child value

methods. The present invention is thereby able to provide automatic value inheritance upon child creation.

Those skilled in the art will appreciate that various adaptations and modifications of the just-described preferred embodiments can be configured without departing from the scope and spirit of the invention. Therefore, it is to be understood that, within the scope of the appended  
5 claims, the invention may be practiced other than as specifically described herein.

100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000